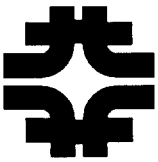

Fermi National Accelerator Laboratory

FERMILAB-Pub-94/033

Comparisons of CPS Performance Under Different Network Loads

Aleardo Manacero, Jr.

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

dcce/ibilce/UNESP
S.J.R. Preto, Brazil

January 1994

Submitted to *IEEE Transactions on Computers*

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

Disclaimer

Comparisons of CPS performance under different network loads

Aleardo Manacero Jr.*
dcce/Ibilce/UNESP - Fermilab

Abstract

This work presents results from the comparison tests done over two different networks using the CPS [1] (Cooperative Process Software) distributed tool. One system was a non-dedicated UNIX [2] cluster composed by several RISC workstations. The other system was another UNIX cluster which is dedicated to run CPS exclusively for a single user. Benchmarks were obtained through the submission of a simple test program over the cluster using three different approaches: a single process task, a CPS synchronous process and a CPS asynchronous process. The speed-ups achieved in both systems were then compared in order to make some remarks about the efficiency of CPS under different conditions of use. From these comparisons it was possible to conclude that the CPS should be improved to run in a non-dedicated cluster, like the UNIX cluster used in this experiment is.

Index terms: CPS - Cooperative Process Software, dedicated network, distributed processing, multi-user network, network traffic, performance analysis.

1 Introduction

The use of several workstations to perform a single task instead of a single workstation would increase the speed of problems resolution dramatically. This could be antecipated from the description of the CPS management system [1], a software tool designed to manage parallel processing over parallel or distributed machines, which points to a big gain if your process could be split into several subprocesses. The reconstruction programs of particles physics have this characteristic.

Experiments such as the E-791 collaboration at Fermilab [3, 2] collect large amounts of data from the collision of tagged particles in a fixed target detector. This data is composed of events, which are collections of electric signals originated in several sections of the detector (Cerenkov, drift chambers, calorimeters, etc.), and stored in some kind of storage device to be analyzed at a later moment [4]. The analysis process starts with the reconstruction of such data, that is, the search for the vertexes and trajectories of all particles that appear in a single event. Since all events were stored in a sequential order, it is possible to treat each of them as an independent entity, which could be examined by the reconstruction code without any information from another event. This characteristic makes it appropriate to run in computer farms, under tools like CPS.

Since the use of farms significantly improves the speed of the reconstruction and analysis efforts, it is also interesting to know the behavior and the performance of the CPS when it is applied to a non-dedicated network of workstations, which could provide another source of computing power. This performance also could be a good measure of the network performance when operating under a distributed process problem since the parallelism offered by the CPS can be understood as a distributed operation over the system and such an architecture (the network) is quite different from the dedicated farm architecture [5].

In this work one of the E-791 programs for data preparation was used as the benchmark program. This program was chosen for the ability to make its CPU demands per data byte range from some milliseconds to a few seconds, which was a necessary condition to the benchmark program. The following section describes the two systems tested in this comparison. In the third section the tests on a dedicated IBM test farm containing six processing nodes are described. A fourth section describes the tests on the UNIX cluster in the Physics Department. Following these are sections discussing the results from both tests and general considerations.

2 Computing environments

This work was executed on two different computing systems. The first and more recognizable one is a conventional network of workstations assembled in a multi-level bus topology. The other system is a computer farm, which is a network of workstations assembled in a way that resembles a master-client parallel architecture.

*currently at the State University of São Paulo - UNESP
Dept. of Computer Science and Statistics
R. Cristovão Colombo, 2265
S. José do Rio Preto, 15055-000, Brazil.

Figure 1 shows the conventional network, where the most important characteristic is the fact that it is a multi-user environment, with no-restrictions in its usage. It has cross-mounted disks, which turns the network traffic into a very influential factor in the system performance. Those conditions represent a very common pattern in the configuration of networked workstations and indicate the reason for the inclusion of such a system in this comparison.

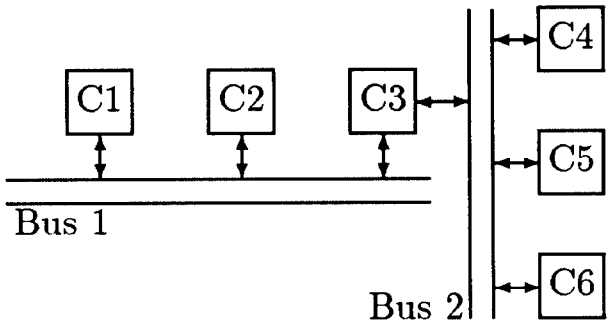

Figure 1. Multi-level bus network.

Although computer farms are not a major concern in production from the industry point of view, they are very useful for the kind of processing done in the reconstruction and analysis of high energy physics (HEP) experiments, such as E-791, where the level of integration among the parallel units is very small since the data can be split into small independent fragments. The farm's basic architecture is shown in figure 2. The system consists of a set of processing nodes (conventional workstations with little memory and disk capacity, fast CPU's and no monitors) whose only function is crunch data at the highest speed possible. The system is usually dedicated to a single user, running only one job at a given time.

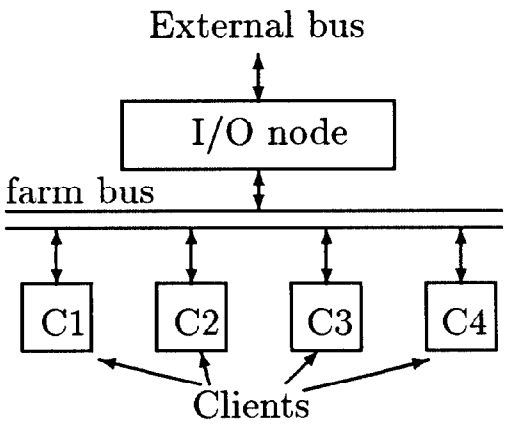

Figure 2. Farm architecture.

The basic differences between both systems are the load applied to each one and the cost of the equipment used. The cost of one farm system is lower than of the conventional one since the farm workstations are configured without some costly devices that are not necessary for the kind of processing to be done. Also the non-CPU system load for the farms is lower, since they are user dedicated and the network traffic is restricted only to the traffic created by the user's job, while in the conventional network traffic is originated by the combination of traffics generated by several users accessing different disks and their respective jobs.

3 The IBM farm performance

The tests on a small IBM farm, only six processing nodes, were done in order to have comparison parameters to analyze the speed-up obtained in the UNIX cluster. In these tests the time spent between the job submission and the job completion, the release time (T_r), was the unique concern and the only parameter measured. CPU occupation, data transfer rates and other usually useful parameters were not measured because they are meaningless in an environment where the CPU and the network are shared with several other users.

The release time was used to calculate the speed-up provided by the parallel runs against the single process run. The single process run consists of doing the complete job using just one CPU without the CPS management. It should be made clear that this task is different from a CPS submission using just one CPU since CPS splits every job into three subprocesses, a reader process, a writer process and a client process, what leads to sharing of the CPU between three processes, plus a job manager created by CPS, instead of only one global process like in the single process run.

Although the code used in these tests could be tuned to any event per second ratio, they were done considering only two opposite conditions, one where the job was I/O limited and another where the job had an execution time close to the real reconstruction time in the E-791 case. From these tests were obtained data of speed-up versus number of CPU's in use. A different study can be made to provide graphical help to decide when a job should be parallelized or not and could be based in the interaction between the time of the single process run, the overhead time introduced by the CPS management per client and the number of clients.

The Table 1 shows the results for the IBM farm in terms of release time for runs over different numbers of processing nodes. These tests were done under the following conditions:

Process	# CPUs	T_r (A)	ev/sec (A)	T_r (B)	ev/sec (B)
Single	1	43	34.884	1247	1.203
MP Async	2	45	33.333	1405	1.068
MP Async	3	48	31.250	697	2.152
MP Async	5	55	27.273	350	4.286
MP Async	7	61	24.590	229	6.550

Table 1- Events/second with the asynchronous CPS on an IBM farm.

Case	1 client	2 clients	4 clients	6 clients
A (35 ev/sec)	0.956	0.896	0.782	0.705
B (1.2 ev/sec)	0.888	1.789	3.563	5.445

Table 2- Speed-up for asynchronous CPS on an IBM farm under different processing time conditions.

1. Number of events examined in each run: **1500**
2. Events/second in single process: **(A) 35 ev/sec** and **(B) 1.2 ev/sec**

From this table it is possible to make a second table showing the speed-up provided by the parallel runs over the farm when compared to the single process run. This speed-up can be seen in the Table 2, where the first row is the speed-up for an I/O bounded task while the speed-up for a CPU bounded task is in the last row, and in Figure 3 at the end of this text.

Looking at these two tables, especially in the speed-up table, one observes that parallel execution only gains if the task requires enough processing to not be I/O-bound. That is, if the task has a predominance in I/O operations when compared with CPU operations, there is no advantage in using several CPUs for its processing. The amount of CPU load that is necessary to be optimal for a certain number of nodes or, from a different perspective, how many nodes are adequate for a certain CPU load, has not been analyzed.

One interesting point to be noted is the evident anomaly of the parallel runs in the I/O-bound case, where the speed of processing decreases with the addition of new CPUs. This fact occurs because the activity of reading events took more time than analyzing events. Since CPS activities to manage CPUs are directly proportional to the number of CPUs, for more CPUs CPS takes more time for itself and the system performance will degrade.

However, since in a CPU-bound execution there are clear advantages in using CPS and farms, with speed-ups close to the expected optimals, one could expect that the use of CPS in a regular network of workstations should provide responses that, at least, encourage its use in such systems as well. Unfortunately it will be shown in the next section that this is not true for most ordinary networks, with architectures not dedicated to a specific task.

4 The UNIX cluster performance

Since the main interest here is the test of the CPS management system in a regular network, the same tests that were performed in the IBM farm were applied to the physics department cluster of SGI machines. Its load is almost constant during daytime and does not drops significantly during night time, since most of the users leave batch jobs to be run after hours.

The efficiency of the system should be similar to that provided by the farm, because when the CPS strategy is combined with the characteristics of the event reconstruction algorithm one could antecipate that if n workstations are used to perform the parallel code there will be a speed-up close to n. So, if the user assemblies $n+1$ CPUs, leaving one of them to write and read data, as it is done in the farms, he/she gets n CPUs doing the job. Since no one of them needs to know what its neighbor is doing, he/she has independent tasks performed by independent CPUs. Therefore if there are n CPUs in the system it should do the work in a n-th part of the time spent by just a single CPU doing the same job, excluding the general I/O time of course. It must be clear at this point that the cpu load does not have a great influence in these results since all nodes have the same cpu usage and the speed up is relative to the single process that is also running in one of those nodes.

4.1 I/O bound tasks

Because it is known that I/O-bound tasks do not have any improvement in their resolution time when run in parallel processors it is useful to have a benchmark using such tasks since this can provide some indication about the dependence between the network traffic and the distributed processing performance. The E-791 stripping code was used as an I/O bound task. It consists of reading an event and selecting it or not based on some limit values taken from the event data structure and geometry pattern data. This particular job reflects the interest in I/O bound tasks, since the stripping operation over the primarily reconstructed data set does not claim much CPU time and the reading and writing operations are performed over 8mm tapes, which are not fast.

The Table 3 contains the results of several runs of the stripping job over the same input data used in the farm tests, and it was obtained under the following

conditions:

1. Server node: **one Personal Iris 4D/35 workstation**
2. Number of events examined in each run: **12971**
3. Events/second in single process: **49 ev/sec**, which implies in an I/O-bound limitation

These results imply in the speed-up relations shown at Table 4, where the first row shows the speed ratios among the CPS synchronous runs and the single process run. The second row indicates the speed ratios among the the CPS asynchronous runs and the single process run. The third and fourth rows show the speed-up among the several synchronous runs and the CPS run over just one CPU (fnpx05) and the performance increase when we use an asynchronous approach instead of the synchronous one, respectively. Figure 4 shows the plots of these speed-up curves.

To understand these numbers it is necessary first to define the differences between the synchronous and asynchronous versions of CPS. The differences are in the interprocess communication mechanism used in each. The early versions of CPS had only the synchronous mode, where each time that one process had to transfer a message to another process it had to do so synchronously, stopping its activities while its partner was sending/receiving that message. In order to get a better performance, CPS was changed to enable an asynchronous approach to communication between processes. This approach was used in this case to make the data transfer between the reader process and all the clients. With this structure the reader can send a packet containing data and resume its reading operation even before there is a client looking to read that packet. The same was not done for the communication between writer and client processes because the speed gain achieved is minimal (much less than 1%) and also due to some safety protocols used to guarantee the processing and the statistical recording of the complete data set.

One can see from table 4 that CPS is not effective on tasks that are I/O limited since none of the different configurations achieved a performance better than the single process case. This is an interesting result for anyone doing a project's computer configuration planning, since it implies that tasks such as the one performed by the stripping code should not be run in a farm or in any parallel environment unless it becomes a CPU-bound task.

Another remark about these results is that the performance of the asynchronous version does not change much for different number of working nodes while the same does not hold true for the synchronous case. The speed-up of synchronous CPS has a huge variation with the number of CPU's in use, with processing speeds ranging from near 15% of the single process speed with one CPU to more than 90% for five CPU's, while the range for the asynchronous CPS ranged from 85% to 98%. The communication protocol plays a large role in those numbers and is the fundamental culprit in the low efficiency of synchronous CPS since in this case the reader process cannot concurrently collect data from the tape and each client must explicitly indicate that data is needed. This causes a serious delay in the reading activity since the reader must wait for the clients get the data just read to be able to read further data from tape. This delay is minimized when new clients are added to the system, because the reader could have shorter waiting times while serving more nodes. In the asynchronous case the waiting times are minimal because each client serves itself with buffered data, so the reading process could run at a tape speed, except for overhead introduced by the network traffic. Then, its performance is more constant independent of number of nodes.

A resource allocation policy could be guided from the "execution time x number of nodes" curve. In the synchronous case this curve shows more clearly how an excessive number of nodes can deteriorate the system performance. This deterioration comes from the fact that the above a certain level of parallelism, both the

Process	# CPUs	Elapsed time	events/second
Single	1	271	47.863
MP Sync	1	1966	6.598
MP Sync	2	882	14.706
MP Sync	3	388	33.430
MP Sync	5	294	44.119
MP Sync	7	328	39.546
MP Sync	9	373	34.775
MP Async	1	277	46.827
MP Async	2	276	46.996
MP Async	3	307	42.251
MP Async	5	291	44.574
MP Async	7	315	41.178
MP Async	9	316	41.047

Table 3- Events/second in a I/O-bound case for single process.

# of processors	1	2	3	5	7	9
Sync/SP	0.14	0.31	0.70	0.92	0.83	0.73
Async/SP	0.98	0.98	0.88	0.93	0.86	0.86
Sync/Sync	1.00	2.22	5.07	6.69	5.99	5.27
Async/Sync	7.09	3.20	1.27	1.01	1.04	1.18

Table 4- Relative speed-ups achieved in the I/O-bound case.

overhead given by the interprocess communication and the processes' waiting times are big enough to have always more processes waiting for data than data available. The same conclusion can also be drawn from assynchronous CPS, although in this case the optimal number of parallel processes is even smaller than for synchronous CPS due to its better communication mechanism, which reduces the overhead to proceed data transfers.

4.2 CPU-bound tasks

The experimental data obtained for the execution of the CPU-bound task is presented in Table 5. This kind of task represents the reconstruction code, which is a cpu intensive process. In order to perform this test with the same data pattern that was used in the I/O-bound task tests, a slightly modified stripping code was used instead of the regular reconstruction process. This was the same code used for the tests on the IBM farm, what turns feasible the comparisons between both systems. The modifications needed in the stripping code were minimal, the addition of a loop in order to delay the event selection by an adjustable amount of milliseconds. The conditions for this test were:

1. Server node: **one Personal Iris 4D/35 workstation**
2. Number of events examined in each run: **1286**
3. Events/second in single process: **5 ev/sec**, which implies an event-dominant case

Table 6 is the speed-up table analogous to table 4, showing speed ratios between the asynchronous and synchronous CPS versions and the single process run. Figure 5 shows the speed-ups achieved in this experiment.

From these tables we see an interesting difference between the asynchronous and synchronous versions of CPS. As seen in table 6 the asynchronous version provides better speeds than its counterpart. This is true for the configurations tested, where the minimal improvement was 13%, but its curve (figure 5) indicates that the difference decreases when the number of nodes is increased. In theory this difference shall approach zero if enough nodes are added to the system.

The explanation of why this occurs is based on I/O-bound task limitations, since if the tape dictates system speed, the communication protocol cannot play a large role in the efficiency, as seen in the previous study case for a larger number of CPU's. So, if the number of nodes linked to the real processing grows, the number of events processed by each one of them decreases,

Process	# CPUs	Elapsed time	events/second
Single	1	255	5.026
MP Sync	1	1113	1.155
MP Sync	2	1065	1.208
MP Sync	3	665	1.934
MP Sync	5	289	4.460
MP Sync	7	218	5.899
MP Sync	9	200	6.430
MP Async	1	882	1.458
MP Async	2	858	1.499
MP Async	3	509	2.527
MP Async	5	252	5.103
MP Async	7	185	6.951
MP Async	9	177	7.266

Table 5- Events/second in an event-dominant case.

# of nodes	1	2	3	5	7	9
Sync/SP	0.230	0.240	0.385	0.887	1.174	1.279
Async/SP	0.290	0.298	0.503	1.015	1.383	1.446
Sync/Sync	1.000	1.046	1.674	3.861	5.107	5.567
Async/Sync	1.261	1.242	1.306	1.144	1.178	1.130

Table 6- Speed-up achieved in an event-dominant case.

making them wait for the reader more frequently than if there are few processors. That represents an I/O-bound like task.

In the other hand, neither CPS versions had shown a significant improvement in the processing speed, with a maximum gain of less than 50% when compared to the single process run but using a total of 8 processing nodes. This results means that the network traffic is dominant in the determination of the system speed. Since the network configuration enables users other than CPS processes it creates a significant amount of strange traffic competing for the single physical support for communication, which implies bigger delays in interprocess communication and, therefore, a slower system for the CPS process.

5 Comparisons between configurations

The comparison of CPS running over a farm and over a non-dedicated network can be divided into two different aspects: the computational effort per data byte and the network load influences on system performance. The former will not be discussed in detail here because it is not the main topic of this work. It is enough to point out that the parallel systems do have a very poor performance when the computational effort to process the data is much smaller than the effort spent doing input/output operations.

This does not condemn parallel schemes for data ac-

quisition for example, that are mainly dedicated to input/output operations, because in these schemes the relevant aspect is the speed in which data can be collected and/or stored. In such systems the acquisition and storage of the data is what is really done in parallel while the processing of this data in order to store it might or might not be done in parallel. In any case here the reason to use parallel systems is the speed of data acquisition, which means that the ratio between the time spent processing a given amount of data and the time spent doing I/O for this same data is favorable to the parallelization of such activities.

With respect to the network load, there are some interesting results to be pointed out. Most of them are related to the efficiency of CPS management over a non-dedicated network, which was well below to that expected for the kind of parallel algorithms and data used here. Some deficiencies were already presented in the previous sections but will be presented in detail here.

The performance of CPS in a dedicated environment such as a farm shows a speed improvement nearly optimal; a highly cost-effective system. But the same does not hold when CPS is applied to a more general network, where there is no exclusiveness in the use of the nodes. In this situation the performance drops significantly, achieving no more than a small gain relative to the speed of the sequential execution in the single process scheme.

Table 7 lists the ratios (in percentage) between the theoretical speed-up, given by the number of effective processing nodes, and the experimental speed-up achieved for each configuration tested, given by tables 2, 4 and 6. Lines 1 and 4 give the values for the IBM farm, the second and fourth line give them for the UNIX cluster running synchronous CPS while the lines 3 and 6 contain the values for the CPS asynchronous runs in the UNIX cluster. The first 3 lines are related to I/O-bound tasks while lines 4 to 6 are related to CPU-bound tasks. The same data is plotted on figure 6.

Before start the analysis of the CPU-bound case it is necessary to make a remark about the strong similarity among the three I/O-bound cases. There, the percentages achieved are very similar to each other when there are more than 2 nodes in use. The same does not hold for the CPU bounded tasks, where the numbers from the farm experiment are widely different from the others.

When the task is CPU-bound the performance of the farm environment is far better than the performance of the network environment. The reasons for such differences were already mentioned and are linked to the extra traffic introduced by additional messages, i.e.

Task type	Environment	1	2	4	6	8
I/O bounded	farm	95	45	20	12	NA
	UNIX synch.	31	35	23	14	9
	UNIX asynch.	98	44	23	14	11
CPU bounded	farm	89	90	89	91	NA
	UNIX synch.	24	19	22	20	16
	UNIX asynch.	30	25	25	23	18

Table 7- Percentage of optimal speed-up achieved in different environments and tasks for some maximum theoretical speed-ups.

messages that are not related with the CPS job, competing for the use of the network physical layer in the UNIX cluster.

But all three environments showed a reasonable steadiness in their experimental/theoretical speed-up ratios along the changes in the number of working nodes in the CPU-bounded cases. This steadiness is remarkable for the farm tests, where the variation observed between those ratios were inside a $\pm 1.4\%$ interval from the average value. For the non-dedicated network the results were less steady but also kept some linearity along the tests ($\pm 25\%$ from the average value) relative to I/O-bounded cases (more than $\pm 100\%$ fluctuation from the average).

This steadiness can be explained by the fact that if you can keep all the processors busy the improvement in the system speed will be proportional to the increase in the number of processors in use. This also explains the increased fluctuations observed for the conventional cluster, where the cpu usage dropped significantly due to the extra traffic in the network.

Although the steadiness present in the farm experiment with a cpu-bounded task is, in itself, interesting, it also leads to the conclusion that there must be a threshold in the number of worker nodes in use, since if too many nodes are used for the same amount of data, they are going to be idle part of the time, waiting for more data to process, as per the I/O-bound case. When the number of nodes is above this level the system does not provide any gain per additional parallel node. The determination of this threshold is a task that should be done prior to the farm installation, saving processing time and avoiding extra costs to the user.

6 Conclusions

Among the conclusions not related with CPS that can be drawn from this experiment are two especially important ones that are interesting to both, the system

manager and the final user of the system, for different reasons.

The first one is the fact that not all jobs are suitable for parallel processing. The user can use this information to plan and choose which tasks are suited to parallelization and which are not. The manager has to use this information to avoid assigning parallel resources to users whose tasks are not suited for that.

An extension of this is the fact is the second conclusion and it is related with the level of parallelism that each task admits. Although there are jobs that are not parallelizable at all, there are some others that are parallelizable only up to certain levels (in reality all jobs can fall in this category if you consider one node as a system with parallelism level zero) and this is a characteristic that should be considered when considering the use of parallel systems.

Determination of the parallelism level adequate for a given kind of task is very useful to the user, who can plan (and request) a computer configuration that is, at the same time, optimal in speed and cost. This shall give him a more consistent argumentation towards the obtention of his needs. On the other side the manager could use the same kind of information to justify the denial of any number of nodes that exceed the optimal level for a specific utilization.

A few remarks about the use of CPS under different situations: the major conclusion is that CPS processes cannot afford competition for the resources in use, that is, CPS, as well as other "parallel processing" tools, does not do well sharing the hardware resources with other users. In the CPS case the sharing of the network is the major cause of poor performance when compared to an exclusive access environment, as it was demonstrated by the results in section 4. Sharing the cpu did not play a large role here since the parallel results were compared with sequential runs processed in the same condition, i.e. various users sharing the cpu.

However, if the access to all of the system resources can be restricted to a single user which runs the parallel process, CPS becomes a very useful tool, since it is simple to use and gives a performance that is reasonably close to optimal, introducing only a small overhead due to synchronization and communication management processes.

In conclusion, the performances of CPS processes indicated that it is useful for dedicated systems but must be improved for use in a multi-shared environment. However CPS was developed having in mind that the user is a physicist and the problem to be solved is one that involves large calculations over large amounts of data and that this could be done in a dedicated system. The trials done here were investigations trying to verify if the scope of applications for CPS can be augmented with no additional development effort. This proved to be untrue.

7 acknowledgements

This work was performed with the support of the Fermi National Accelerator Laboratory and the State University of São Paulo - UNESP. The technical support received from the Computing Division and the Physics Department of Fermilab were very helpful to the completion of these experiments.

References

[1] M. Fausey et all; "CPS & CPS Batch Reference Guide - Version 2.8", Fermilab, 1993.

[2] R. A. Sidwell; "E791 status report", 7th Meeting of APS, Division of Fields and Particles, vol1., p.672-674, 1993.

[3] J. Appel;"Hadroproduction of charm particles", Annu. Rev. Nucl. Part. Sci., v.42, p.367-399, 1992.

[4] S. Amato et all; "The E791 parallel architecture data acquisition system", Nuclear Instr. and Meth. in Phys. Res. A, n. A324, p.535-542, 1993.

[5] R. Hance et all; "The ACP Branch Bus and Real-Time Applications of the ACP Multiprocessor Systems", Fermilab-Conf-87/76, 1987.

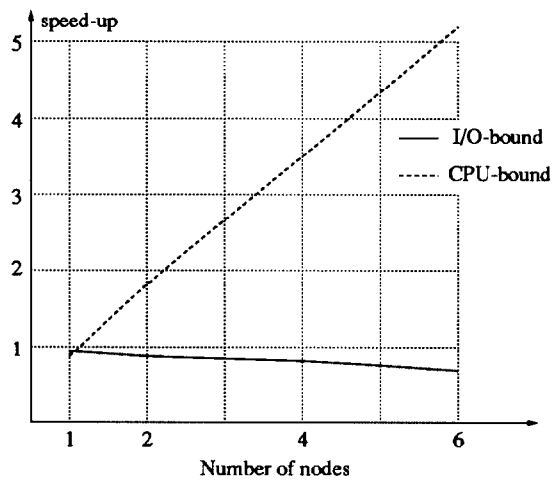

Figure 3. Speed-ups achieved in the farm system.

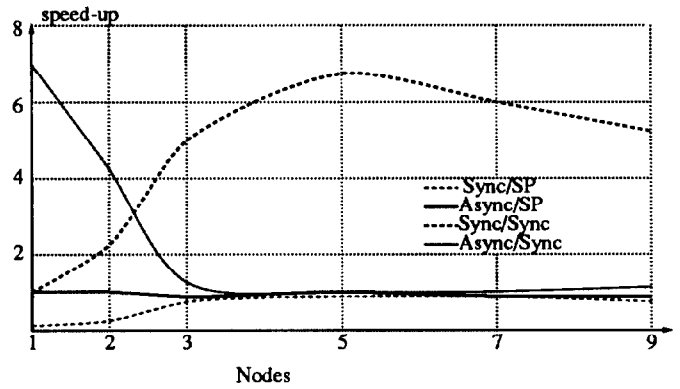

Figure 4. Speed-ups achieved in the UNIX cluster with an I/O-bound task.

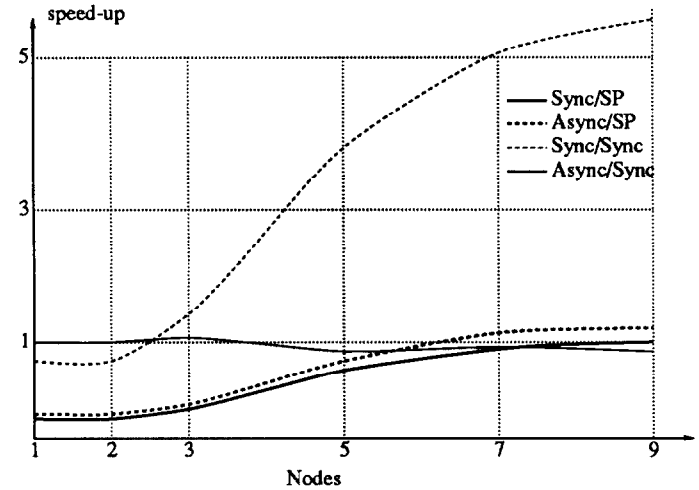

Figure 5. Speed-ups achieved in the UNIX cluster with a CPU-bound task.

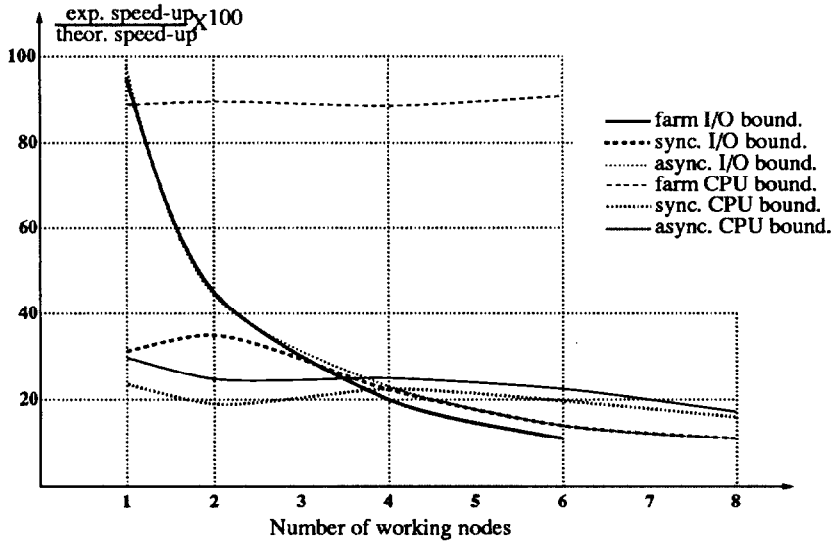

Figure 6. Relative speed-ups achieved in relation to the theoretical optimal speed-up.